Mobile application platform selection

University of Oulu
Faculty of Information Technology and
Electrical Engineering
Bachelor's Thesis
Joonas Nygård
12.5.2019

Abstract

Native and web apps have their own advantages and disadvantages in the field of mobile app industry. This fact has forced industry to make reforms and develop new tools and technologies to mitigate the disadvantages by both platform types. Different cross-platform development approaches have lowered the costs of developing apps to multiple different platforms and progressive web apps (PWA) have improved efficiency and user experience for web apps. This thesis strives to clarify the selection, which platform or approach the company should choose for their upcoming app. This is done by finding the properties and requirements found important by shareholders and finding out how capable platform/approach is meeting with the properties.

Keywords
platform, native, web, cross-platform, PWA, mobile app, development

Supervisor
Lecturer, Phil. Lic., Antti Juustila

Glossary

API	Application programming interface
CLI	Command-line interface.
Cross-compiled app	Cross-platform mobile app which is compiled separately for each desired platform.
Cross-platform app	Mobile app developed with tool allowing development for multiple platforms at once.
CSS	Cascading Style Sheets. Standard markup language for describing HTML presentation.
HTTPS	Hypertext Transfer Protocol Secure.
HTML	Hypertext Markup Language. Standard markup language for creating web applications.
Hybrid app	Cross-platform mobile app developed with web technologies and embed inside a native container.
Interpreted app	Cross-platform mobile app which is interpret for each desired platform.
Mobile app platform	Platform for mobile applications. Web or native platforms.
Native app	Mobile app developed for some particular mobile platform like iOS or Android.
PWA	Progressive Web App. Web app with extra functionalities that were before available only in native apps.
SDK	Software development kit
UE	User engagement.
UI	User interface.
UX	User experience.
Web app	Mobile application developed for web platform.

Contents

1. Introduction

Use of smart mobile devices has increased in the 2010's that much, most web traffic is produced by mobile devices. Because of this, web-developers were forced to take the mobile users into account and start developing the web mobile-first. Modern web-development uses technologies like frontend frameworks that help developing **web apps** that scale on different size screens and devices. Developing web app rather than native app for smartphone is also much cheaper and reaches larger amounts of devices and people (Charland & LeRoux, 2011; Han Rebekah Wong, 2012; Xanthopoulos & Xinogalos, 2013). Web apps have still it's drawbacks compared to native apps, so it's not always sensible choice (Opinion: Native vs. mobile web app - are we missing the point?2012; Charland & LeRoux, 2011).

Native apps are those applications that user downloads and installs to mobile device from app marketplaces (e.g. Google Play or Apple's App Store). Native apps were the primary way to develop and release mobile apps to market. When developing native apps, you can get much more out of the device hardware than web apps. Native apps can access basically all the hardware features that a device has. They also usually have better usability/user experience, because they can be in many ways more efficient than web apps (Kim, 2013).

Different **cross-platform** approaches have tried to resolve the issues of costly development of native apps for multiple platforms like iOS and Android. (Bai et al., 2019). We are taking a closer look towards three different cross-platform approaches: **Hybrid apps**, **Interpreted apps** and **Cross-Compiled apps** (Ciman & Gaggi, 2017). All these approaches strive to reduce the workload of producing native app for multiple platforms, but the way they implement it, differs a lot. Many factors like device sensor access limitations, user experience and performance differences should be taken in concern when selecting cross-platform framework for developing a new app. We will be comparing the abilities of these approaches and tools.

The most recent advancement for web platform has been **PWA** (**progressive web-apps)** that implement many features of native mobile apps (e.g. push notifications and working offline) but still actually operates on web-browser. User can also add the app to devices home screen, and after that, PWA can act like a native app on user's phone. User has icon at the devices home screen and the app opens in a completely own window rather than a tab in devices web browser. This gives better user experience and raises the probability that the user will keep using the app in future (Kho, 2018).

Making decision, what platform to choose for upcoming mobile application can be hard to make. Web apps reach more people with smaller investment, they do not require installation and can be visited quickly, but native apps enable greater quality, wider technical possibilities, better efficiency and have greater commitment by user (Nakajima, 2012). Many factors should affect developer decision, do rather develop just a web-app, native app, both or maybe use some cross-platform approach. I will set two research questions and answer them, so that making the decision for upcoming mobile app platform would become easier:

RQ1: What factors or requirements should affect the mobile application platform /approach selection?

RQ2: How well different platforms and approaches perform on different requirements?

Rest of this thesis is organized in following order; Chapter 2 goes through the research methods. Chapter 3 looks in the prior literature. Chapter 4 presents findings by comparing platforms and approaches. Chapter 5 discussion and conclusion.

2. Methods

In this thesis I used literature review as the research method. Search for prior literature was done mainly using Scopus but also EBSCO, Google Scholar, IEEE Xplore and Oula-Finna were used. Also, Google Search were used to find documentations for cross-platform frameworks. RefWorks ProQuest were used to manage references. Searches described in following sections were made in Scopus. This chapter describes how my research proceeded and thinking evolved also justifying the changes I decided to make after gaining more knowledge towards the subject.

2.1 Initial search

Initial research started with simple query "TITLE-ABS-KEY (web AND vs AND native AND mobile AND apps)", which gave only four documents in which 3 were relevant for my research. After this I used more sophisticated search query; "TITLE-ABS-KEY (native AND mobile AND (app OR apps) AND mobile AND web AND (app OR apps))" which gave 170 documents. Unfortunately, many of these documents were irrelevant for my purposes. After this I evolved my search query in the following form: "TITLE-ABS-KEY (modern AND mobile AND development AND native AND web AND (app OR apps))" This query gave me just 12 documents, some of which were useful. However, the most successful search query giving many relevant documents was following: "TITLE-ABS-KEY (web AND native AND app)" giving 151 documents, many of which were interesting and useful for my research.

2.2 Clarifying guidelines

After gaining understanding towards the subject by reading articles found, I found out that PWA: s (progressive web-apps) and hybrid apps were essential topics as well and I should include them in my research. Search for PWA with query "TITLE-ABS-KEY (progressive AND web AND apps)" in Scopus gave just 24 documents, since it's still new subject. Most of the results were from year 2018. Search for hybrid apps with query "TITLE-ABS-KEY (hybrid AND apps)" gave 296 results, of which I chose few articles into closer review. In this point I noticed that it could be useful to take more closer look towards hybrid apps, and also other cross-platform approaches in future research.

For practical reasons I divided prior literature in 3 subclasses while working with them. Class no 1. contained literature comparing web and native apps. 2. hybrid apps and cross-platform app development and 3. progressive web apps. Organizing these topics into subclasses helped me focusing for literature dealing with one topic at the time.

2.3 Expanding research

I understood that I should also discuss about the other cross-platform approaches than just hybrid apps. These other approaches differed fundamentally from hybrid approach so it would be necessary to include them in the research and compare them. I started searching related literature with query TITLE-ABS-KEY (mobile AND cross AND platform AND development) which brought 701 results. I sorted documents by relevancy and selected six interesting articles to read. After reading these articles discussing and comparing different cross-platform approaches I decided to take a closer look towards

interpreted and cross-compiled approaches. I also felt that it would be useful to take a closer look towards at least one actual framework for each category, so I started searching documentations from Google for frameworks that were discussed in the research papers I read. I found out that some key frameworks discussed in the papers were discontinued with other frameworks replacing them, so I decided to explore these new popular frameworks instead.

2.4 Overview

Overall, I found more than 30 interesting articles which I divided into 3 categories. Fourteen articles discussing **cross-platform approaches** (Boushehrinejadmoradi, Ganapathy, Nagarakatte, & Iftode, 2016; Ciman & Gaggi, 2017; Dalmasso, Datta, Bonnet, & Nikaein, 2013; Dhillon & Mahmoud, 2015; Ebone, Tan, & Jia, 2018; El-Kassas, Abdullah, Yousef, & Wahba, 2014; El-Kassas, Abdullah, Yousef, & Wahba, 2017; Martinez & Lecomte, 2017; Nunkesser, 2018; Palmieri, Singh, & Cicchetti, 2012), seven articles discussing and comparing **native and web approaches** (Opinion: Native vs. mobile web app - are we missing the point?2012; Charland & LeRoux, 2011; CLABURN, 2014; Han Rebekah Wong, 2012; Liu et al., 2015; Ma, Liu, Liu, Liu, & Huang, 2018; Nakajima, 2012) and six articles discussing **Progessive Web Apps** (Fortunato & Bernardino, 2018; Frankston, 2018; Gronli, Hansen, Ghinea, & Younas, 2014; Kho, 2018; Luntovskyy, 2018; Shahzad, 2017). Also documentations for different cross-platform development frameworks were used to make a closer comparison about the different frameworks (Documentation - apache cordova.; Getting started · react native.; Titanium SDK - appcelerator platform - appcelerator docs.; Xamarin documentation - xamarin.).

3. Background

Purpose of this chapter is to gain basic understanding in the subject by looking into prior literature discussing and comparing different platforms and approaches in mobile app development. Intention is to find all reckoned platforms and approaches and find advantages and disadvantages related to them.

3.1 Fundamentals of native and web apps

The two main types of applications are the native and web applications. PWAs and hybrid apps are kind of combinations of these two, trying to use the best sides of both. There are two mobile OS leaders for native app platforms, Android and iOS. If developer want to publish a native app, he must create app at least for these two platforms (Charland & LeRoux, 2011; Nakajima, 2012). Usually Android applications are developed in Java or Kotlin languages, and iOS apps in Objective-C or Swift. Web apps instead are developed to run on any device, which can run any modern web browser. Web apps are usually developed with some modern frontend frameworks like ReactJS, Angular or VueJS, but in the end, they all compile to HTML, CSS and JavaScript, which are the standards supported by browsers (Ma et al., 2018; Nakajima, 2012).

There have been number of studies considering differences between web and native apps. There are major advantages for both platforms. For web apps e.g. cost of development and cross-platform compatibility and for native apps performance and wider access to device features (Charland & LeRoux, 2011; Gronli et al., 2014). Many times, the decision is made completely due to financial aspects, e.g. if company wants to produce app in the cheapest way, or they do not have the recourses to invest more. It is still recommended to inspect more closely what are the differences and advantages between different choices (Opinion: Native vs. mobile web app - are we missing the point?2012; Ma et al., 2018).

3.2 Performance differences and UX

In most cases, reason why native apps performance is the key property winning over web apps, is that lack of performance affects heavily on user experience and due that to user commitment using the app. Native apps can thus feel more responsive and smoother to use (Charland & LeRoux, 2011).

Studies about performance differences and factors behind these differences between native and web apps have shown that native apps in most cases are more efficient and work faster, than web apps (Liu et al., 2015; Ma et al., 2018). Both, native and web apps commonly use some web service to access data. E.g. social media apps could receive the posts and messages via REST (Representational State Transfer) API (Application programming interface). App performance at network level is one important issue, especially in apps, that rely heavily on sending and receiving data from the internet (Ma et al., 2018).

Study by Liu et al. (2015) comparing web and native apps using RESTful services showed that in some cases web apps can act more efficiently than native apps using RESTful services under the same context. E.g. GET operation performance for web apps approached or even exceeded corresponding native apps performance and some POST and DELETE operations performed even better for web apps than native. Still in overall

web apps performed weaker compared to native apps, because they consume much more network traffic and require longer response times (Liu et al., 2015; Ma et al., 2018).

Reason for web apps losing in performance were that network connections differed much more for web apps for same features because web apps need to download resources like CSS and images, which defines the layout. Solution for reducing traffic and requests over network would be proper cache mechanism. If web apps would have optimised caches, we could save up to 80 percent of the requests made, and performance of web apps would come much closer to native apps in terms of network connections. One weakness for web apps was also that web browsers allow only limited number of connections simultaneously to single host, which consumes longer response times, when native apps do not have this restriction. This and previous problems and weaknesses regarding performance of web apps using RESTful services, could be solved with a proper cache mechanism and network optimization (Liu et al., 2015; Ma et al., 2018).

3.3 Development cost differences

Cost seems to be naturally the big question for many companies. Web apps are cheaper to produce (Charland & LeRoux, 2011), but in some cases, in the end it would be more affordable to produce native app. However, developer should have good reason for developing native app, because the cost differences can be vast. The main reason for this is that developer must implement the app individually for all the mobile OS platforms. If developer wants the app to work on two of the most popular mobile operating systems iOS and Android, there is coding work for two individual apps in front (Charland & LeRoux, 2011)

3.4 Cross-platform approaches

Different cross-platform solutions have been developed to decrease the workload developing apps for multiple platforms. There have been discussion about reforming the taxonomy used when talking cross-platform approaches (El-Kassas et al., 2017; Nunkesser, 2018), but in this thesis these solutions are divided in three categories: **Hybrid Apps**, **Interpreted Apps** and **Cross-Compiled Apps** (Ciman & Gaggi, 2017)**.** These approaches and tools perform differently from different requirements like supported platforms, development cost, efficiency, access to device & sensors and UX. Multiple papers (Dalmasso et al., 2013; El-Kassas et al., 2014; Martinez & Lecomte, 2017; Palmieri et al., 2012; Smutny, May 2012; Xanthopoulos & Xinogalos, 2013) have been comparing different cross-platform frameworks by performance, device & sensor access, platform coverage, UX etc. These papers are used to make comparison between different cross-platform approaches for different properties.

3.4.1 Hybrid apps

Hybrid app is app developed using common web technologies HTML5, CSS and JavaScript (Xanthopoulos & Xinogalos, 2013). Then the web app is embedded into native container (WebView) for each platform (Android, iOS etc.). When downloaded and installed, hybrid app includes all components to present the UI, unlike plain web app needs to download HTML, CSS and JavaScript files from server, when loading the page. (Smutny, May 2012; Xanthopoulos & Xinogalos, 2013). Depending on the tool used hybrid approach allows to access device and different sensors, which is not possible with

plain web apps. Main weaknesses in hybrid approach are lack of native UI elements and weak performance, which both affect UX negatively. Most known example of hybrid app framework is **PhoneGap** or the open source version called **Apache Cordova** (Ciman & Gaggi, 2017).

3.4.2 Interpreted apps

Interpreted apps are built with framework, that allows coding the app with languages different from platforms supported languages. When installing the app, also interpreter is installed which is used to execute the non-native code. This kind of approach of course lowers the performance, but its advantage is the opportunity to reuse code written in non-native languages. Example of this kind of tools using interpreted approach is **Appcelerator Titanium** (Ciman & Gaggi, 2017).

3.4.3 Cross-Compiled apps

Cross compiled approach is way to develop cross-platform apps with result closest to actual native apps. Framework enables implementing app with some framework specific language and then generating native app running native-code separately for each desired platform. Although cross-compiled approach does not use any additional layers decreasing the performance while running the app, the generated code cannot reach as good performance as code written by developer, especially in more complex solutions. Examples for cross-compiled approaches is **MoSync** and **Mono**. (Ciman & Gaggi, 2017).

3.5 Progressive web apps

There have been attempts to solve the problems of web apps, like inability to access hardware sensors and lack of efficiency. PWAs have improved these problem areas, making PWA a reckoned alternative to native apps (Kho, 2018). PWAs should have higher performance, better fault tolerance and online activity than traditional web apps. PWAs are developed mainly with same technologies, HTML5, CSS3 and JavaScript and run on web browsers, but PWAs core is so called *ServiceWorkers*, which are responsible for the PWA reforms (Frankston, 2018). As mentioned, web apps consume more traffic and load on network, because the need of downloading UI elements and non-optimized caches. PWA uses more sophisticated caching mechanisms to prevent unnecessary traffic and allowing offline functionality to web apps. PWAs are also required to use cryptographically secured protocol HTTPS for network connections (Luntovskyy, 2018). PWAs act also much like native apps. User can install the app and app icon is added to the phones home screen. When opening app, it will open in completely own window, rather than as browser tab (Luntovskyy, 2018). With all these reforms web-apps seems to take a big step towards being more like native apps.

4. Findings

Purpose of this chapter is to discuss the different requirements and properties considered important by stakeholders involved in the mobile app and compare the different platforms/approaches based on these properties. I'm using requirements and capabilities also used by (Dhillon & Mahmoud, 2015) to compare the approaches (See table 1). The requirements are following: Development (cost), supported platforms, performance, quality of UX, sensor and device access, monetization and updating the app. In section 4.9 we are also taking a closer look towards different cross-platform approaches and within them to actual cross-platform frameworks and comparing their capabilities (See table 2).

4.1 Development

When talking about the cost of development, plain web apps are the cheapest to produce. Mobile web developers are easy to find, and there are number of modern frameworks using common web technologies (HTML, CSS, JavaScript) to make a choice. With a small investment developer can produce/change web app to PWA or make web app to implement parts of PWA that are needed, e.g. push notifications (Kho, 2018). Developing an app with cross-platform framework may vary a lot depending on the tool used. Most of the tools are currently open-source and free to use, but some tools might have hidden costs e.g. due to double licencing or only restricted community edition is free. Anyhow, the idea of cross-platform tools would be that they would lower the costs compared to option where native apps for each platform is implemented separately. More specific comparison between different cross-platform development frameworks is done later in section 4.9 also covering the cost of different tools.

4.2 Supported platforms

Web apps are the cheapest and easiest way to produce an app to cover all devices. Hybrid apps are the best in covering most platforms (web & native platforms). Hybrid app is the most profitable choice in case, it's considered urgent to provide corresponding application for web and native platforms (Smutny, May 2012; Xanthopoulos & Xinogalos, 2013). Sometimes the main reason for developing hybrid app is that plain web apps cannot be published in marketplaces like App Store. Using different hybrid app frameworks, it is easy to capsulate web app into native container, and publish the app as native app. However, some marketplaces have negative attitude towards hybrid apps which are primarily web apps. Apple, for example have in the past declined some hybrid apps from App Store because their development guidelines forbid this kind of direct copies from web apps (Xanthopoulos & Xinogalos, 2013). Closer look towards specific cross-platform tools platform support is done later in sections 4.9. As we can see from the table 2, all the modern cross-platform frameworks support developing apps to iOS and Android, but not all have support for Windows Phone.

4.3 Performance

Because increase of the computing power, efficiency is not as crucial nowadays for all types of apps as it was while back. Approaches based on web technologies (web, PWA, hybrid) are the most inefficient, but valid options, when high performance is not required

(Ciman & Gaggi, 2017; Dhillon & Mahmoud, 2015). These kinds of applications could be e.g. business apps, news apps, social media etc.

Based on research made comparing native apps and apps made with cross-platform tools, native solution is the most efficient type of app, while cross-compiled comes the second (Ciman & Gaggi, 2017; Dhillon & Mahmoud, 2015). So native or cross-compiled apps are the options, when high performance is crucial requirement. Games are usually this type of apps, which require high performance. Performance and efficiency are also factors that can affect heavily on UX an UE (user engagement) negatively if app cannot meet the performance requirements (Charland & LeRoux, 2011; Turgeman, Smart, & Guy, 2019).

PWAs have improved efficiency and UX problems occurred in traditional web apps. Better cache mechanisms prevent downloading unnecessary data, thus making the app run smoother and faster (Kho, 2018; Luntovskyy, 2018). Thus apps based heavily on data retrieved from the internet can be made much more efficient when using PWA reforms like service workers (Frankston, 2018).

4.4 User experience

As mentioned in previous section low performance can affect UX if framework cannot meet the performance requirements. Thus, for achieving good UX, it's also important to be sure that the tool used can produce app that meets the performance requirements.

Another factor affecting UX is the type of UI elements used. When comparing cross-platform approaches, hybrid apps are using web UI elements while interpreted and cross-compiled apps are using native UI elements (Ciman & Gaggi, 2017; El-Kassas et al., 2017). Thus, if native UX is required, native, interpreted or cross-compiled app must be chosen. However study comparing user satisfaction between native and hybrid pointed, that native and hybrid apps had similar ratings on Google Play (Malavolta, Ruberto, Soru, & Terragni, 2015). This result suggests that using native UI elements does not necessarily make better UX than using web UI elements.

4.5 Device & sensor access

Native app or using some cross-platform framework is the way to go, if app is based heavily on different hardware sensors and data, like cameras, microphones, location, or accessing devices files and directories etc. When developing native app for each platform, developer can be sure that (s)he is able to access and utilize all the possible hardware features available for each device. Different cross-platform development tools enable also a good access to different hardware features, depending on the technology used. Usually there are still some limitations for hardware access, when developing hybrid app (Xanthopoulos & Xinogalos, 2013). It's good to make sure from the framework documentation before starting the implementation, that it surely can provide all device and sensor access required. Table 2 comparing tools includes also link to each framework documentation.

The reforms brought by PWA has made it possible to access some important hardware features. If there's no any other special need towards developing native or hybrid app, but a need to access some hardware features like location or ability to for push notifications, PWA might be valid option. Current abilities of web platform can be checked e.g. from whatwebcando.today.

Table 1. Property comparison of platforms and cross-platform approaches.

Property	Web	PWA	Hybrid	Inter-preted	Cross-compiled	Native
Development costs (Producer)	Low	Low	Medium	Medium	Medium	High
Supported platforms (Producer/User)	All devices with web browser	All browsers supporting PWA	Depending on tool (Web and native solutions)	Depending on tool	Depending on tool	Just one
Performance (User)	Low	More efficient than plain web app	Medium	Medium (Interpreter decreases performance)	High (Still not as good as native)	High
UX/usability (User)	Low	Medium	Medium (Web UI elements)	High (Native UI elements)	High (Native UI elements)	High
Access to device and sensors (Producer/User)	Really low	Low	Good (Depending on tool)	Good (Depending on tool)	Good (Depending on tool)	High
Monetization possibilities (Producer)	Unlimited	Unlimited	Possibly limited by framework & marketplace	Possibly limited by framework & marketplace	Possibly limited by framework & marketplace	Limited by marketplace
Updates (Producer)	Easy (Changes appear, when user reloads the website)	Easy	Middle (if user enables automated updates)	Middle	Middle	Middle
Marketplace deployment	No	No	Yes (Possible limitations by marketplace / tool)	Yes (Possible limitations by tool)	Yes (Possible limitations by tool)	Yes

4.6 Monetization & maintenance

If app is planned to be made for commercial purposes, ability for monetization is of course mandatory. Web and PWA apps enable unlimited possibilities (of course within the law) to implement monetization in many ways. Apps business model could be based example on ads, in-app purchases, data selling etc. Monetization in native apps published in marketplaces is restricted with marketplace regulations. Android and Apple developer

documentation (Android developers - monetization., 2019; Business models and monetization - app store - apple developer., 2019) divides monetization options to 1. freemium/in-app purchases, 2. subscriptions, 3. free model/advertising, 4. rewarded products, 5. paid apps and 6. e-commerce. When developing apps with cross-platform tools there might also be some restrictions by the tool used.

Updating native apps or apps created with cross-platform tools installed to device requires permission from the user. User could have disabled automatic updates thus making it more complex to maintain updates on native apps. Web-apps / PWAs are the winner in this category. Maintaining the most recent version is easy for web apps, because updates are performed automatically every time app is launched (Luntovskyy, 2018).

4.7 Overview of properties

Table 1 gathers different properties and platforms/approaches, **web**, **PWA**, **hybrid, interpreted, cross-compiled** and **native** apps based on prior literature (Charland & LeRoux, 2011; Ciman & Gaggi, 2017; Dhillon & Mahmoud, 2015; Kho, 2018; Liu et al., 2015; Luntovskyy, 2018; Ma et al., 2018; Xanthopoulos & Xinogalos, 2013). These papers were comparing some of these platforms and approaches, but none of them compared all of them using these exact properties. I made this compilation mainly by analysing data from these various papers, but some properties were available straight from tables of prior literature. E.g. Luntovskyy (2018) had multiple useful tables comparing web, hybrid and native platforms, which were useful building blocks for table 1.

In table 1 each platform has short verbal description describing how well it manages the property. It suggests platform for every property, e.g. if cost of development is considered the primary factor for app, then web app is the most suitable selection. As we can see from table 1, web and native app has the most winning and losing factors, while PWAs and cross-platform approaches are considered as a moderate compromise solution.

4.8 Cross-platform framework comparison

General comparison of hybrid, interpreted, and cross-compiled approaches have been done in broad level in previous sections, but selection of cross-platform approach or tool cannot be done based on this information. The purpose of this section is to take a closer look into popular open-source frameworks for each category, and compare their capabilities meeting the requirements previously discussed. The four frameworks compared are:

- Hybrid approach: 1. **Apache Cordova (PhoneGap)**
- Interpreted approach: 2. **Titanium,**
- Cross-compiled approach: 3. **Xamarin (Mono)** and 4. **React Native**

Within hybrid approaches, **Apache Cordova** aka PhoneGap and its distributions like Ionic has been the leading framework with good documentation, and thus selected into closer review. For interpreted approach, **Titanium** seems to be the most discussed framework with comprehensive documentation. For cross-compiled approach there have been lot of changes within the past years. Popular MoSync was the leading cross-compiled framework, which have been used in many studies comparing different cross-platform approaches and frameworks. Support and development of MoSync has however discontinued. Instead, there are two promising alternatives using also the cross-compiled

approach, which are **Xamarin** by Microsoft and **React Native** by Facebook (Martinez & Lecomte, 2017). Unfortunately, there's not yet much research made discussing Xamarin (Some on Mono) or React Native. However, each framework has comprehensive documentation, where main information could be gathered.

1. Apache Cordova (PhoneGap)

Apache Cordova is free and open source framework under Apache 2.0 License, for building hybrid apps with **HTML, CSS & JavaScript**. It was released year 2009 as PhoneGap by Nitobi and purchased by Adobe in 2011. Today Adobes commercial distribution of Apache Cordova is called PhoneGap. Also, many other tools for building hybrid apps are built on Apache Cordova e.g. Ionic. Cordova has currently platform support for their development tool CLI for Mac, Windows and Linux only for creating Android apps. To create iOS app developer needs to have Mac and for Windows Phone app developer needs Windows. Cordova enables wide access to device & sensors for android, iOS and windows phone, but as hybrid app it does not have access to native UI elements (Documentation - apache cordova., 2019).

2. Titanium

Titanium SDK is open-source cross-platform framework introduced year 2008 under Apache 2.0 License by Appcelerator for creating cross-platform mobile applications with interpreted approach. Proprietary Appcelerator Studio used with Titanium is paid software, but Appcelerator offers free trial period to try it. Apps created with titanium are implemented with **JavaScript** and then interpreted to native code with interpreter included in the final app. Titanium SDK runs on Mac OS and Windows and supports creating apps for iOS and Android devices. Titanium API has wide access to device & sensors for android and iOS (Titanium SDK - appcelerator platform - appcelerator docs., 2019).

3. Xamarin (Mono)

Xamarin SDK is open-source tool developed by Microsoft owned company named Xamarin. Xamarin is based on open source project Mono to develop cross-compiled mobile apps. Xamarin can be used with VS (Visual Studio) on PC or Mac to create Windows Phone, Android and iOS applications using programming languages C# and F#. Xamarin uses native UI elements and it has good access to device and sensors. Currently use of Xamarin is completely free with VS community edition. Larger companies need to purchase the paid Professional or Enterprise version of VS (Xamarin documentation - xamarin., 2019).

4. React Native

React Native is open-source framework developed and released in 2015 by Facebook. React Native apps are developed with JavaScript and they run especial JavaScript thread in background executing the application logic which is then compiled to native code. Thus, putting React Native to cross-compiled approach category can be debatable. However this thesis uses this classification used by (Martinez & Lecomte, 2017). Although React Native uses web technologies in implementation like hybrid approaches, it uses Native UI elements in the result thus providing better UX than hybrid apps. React Native has documentation providing information about sensor and device access. Some

sensors do not have straight access yet but can be easily used installing external libraries via npm (Node package manager) (Getting started · react native., 2019).

4.8.1 Performance comparison

Performance test by Dhillon and Mahmoud (2015) compared UX and processor and data intensive activities with apps made with cross-platform frameworks from each three categories. PhoneGap was representing hybrid approach, Titanium interpreted approach and MoSync cross-compiled approach. PhoneGap (Cordova) managed to win 2/19 or 11% of performance tests made to it while Titanium won 4/10 or 49% and MoSync won 0/6 tests. There was no clear winner for best performance, but for these three options, Titanium managed best and PhoneGap worst on UI intensive and processor intensive tasks. PhoneGap's average result was also worst and MoSyncs performance was also seen as disappointment (Dhillon & Mahmoud, 2015).

Another important factor measuring apps efficiency is its energy consumption. The battery life of smartphone is not very long yet, so it's important that the app developed is not the one drains the battery fastest. Ciman and Gaggi (2017) studied about battery consumption of apps made with different frameworks. Test app created with MoSync (with JavaScript) performed as the most energy-efficient solution, although it couldn't reach equally good results with true native solution. Other apps made with Hybrid and Interpreted approaches proved to be consuming more energy than apps created with cross-compiled framework (Ciman & Gaggi, 2017).

4.8.2 Comparison overview

Table 2 puts together the basic information for each framework. The data is collected from the framework documentations (Getting started · react native., 2019; Xamarin documentation - xamarin., 2019; Titanium SDK - appcelerator platform - appcelerator docs., 2019; Documentation - apache cordova., 2019) and evaluations made comparing the tools (Ciman & Gaggi, 2017; Dhillon & Mahmoud, 2015).

Apache Cordova is widely used and researched mobile cross-platform SDK, but performance and UX are the weak spots of Cordova. For UI intensive and power demanding apps, Xamarin, React Native or Titanium might be more suitable solution. All the frameworks have good access to device and sensors, but all they have some shortcomings. For most needs the device & sensor access should be enough, but it's good to check from each frameworks documentation that they provide access to those recourses that are needed. All the frameworks also provide support producing apps for iOS and Android platforms but only Apache Cordova and Xamarin have support for Windows Phone apps.

What comes to the development, all the frameworks excluding Xamarin (C#) use JavaScript as implementation language. If there is web development experience within developers involved in implementation, learning these frameworks shouldn't be too time consuming. If developer has experience developing web apps with React, jumping into using React Native wouldn't be a big change because React Native is largely based on React. All the frameworks support development on Mac and Windows, but only Apache Cordova and React Native support Linux. Although all the frameworks are free and open source, there might occur additional costs e.g. due to double licensing. Titanium SDK is free, but it's usually used with Appcelerator Studio, a proprietary paid software. Xamarin

is free for on VS community edition, but larger companies are forced to purchase the paid version. For Apache Cordova and React Native I couldn't find any hidden costs.

Table 2. Mobile cross-platform framework comparison

Framework	1. Apache Cordova (hybrid)	2. Titanium (interpreted)	3. Xamarin (cross-compiled)	4. React Native (cross-compiled)
Developer	Adobe	Appcelerator	Microsoft	Facebook
License	Apache 2.0	SDK: Apache 2.0 Appcelerator Platform: Proprietary software	MIT	MIT
Initial release	2009	2008	Mono: 2004, Xamarin: 2011	2015
Expenses/ revenue model	Free	Using Appcelerator Platform is paid	Large companies need to purchase VS professional / enterprise edition	Free
Documentation	cordova.apache.org/docs	docs.appcelerator.com/platform/latest/#!/guide/Titanium_SDK	docs.microsoft.com/xamarin	facebook.github.io/react-native/docs/getting-started
Supported programming languages	HTML + JavaScript + CSS	JavaScript	C# or F#	JavaScript
Supported development environments (App target platform)	Mac (Android & iOS), Windows (Android & Windows), Linux (Android)	Mac, Windows	Mac, Windows	Mac, Windows, Linux
Device & sensor access	Good	Good	Good	Good (With external libraries)
Native UI elements	No	Yes	Yes	Yes
Performance	Low	Middle	High	High
Supported mobile platforms	iOS, Android, Windows	iOS, Android	iOS, Android, Windows	iOS, Android
Free marketplace deployment?	Yes	No (Appcelerator Platform is paid software)	Yes	Yes

4.9 App categories

Different types of apps usually have different requirements. The core of this thesis was not to deep dive into researching which type of apps have which requirements, but to find out which solutions enables to fulfil different requirements. However, I'm still making a broad classification between fun and functional apps. App stores like Google Play divides their apps to two main categories; games and other functional apps (Google play., 2019). Currently, there are 26 subcategories for functional apps and 17 subcategories for the game section in Play Store. Categories **1. fun apps** containing games and entertainment and **2. functional apps** containing all the other useful apps making our everyday life easier (Opinion: Native vs. mobile web app - are we missing the point?2012).

4.9.1 Fun apps

Within fun apps, properties like user commitment, usability and performance are important, so the better decision in this case might be native, interpreted or cross-compiled app. Games require often high performance and native apps can meet that requirement easier and by this offers also better UX. Also, for fun apps like games, user do not usually have any other reason to use the app, but to spend time and entertain themselves. That is why it is especially important, that user has easy access to the app by home screen whenever he/she has free time. Also, the places where mobile users are looking for games are marketplaces like App Store and Google Play. Popularity of mobile games is largely based on rankings on the marketplace listings. Thus, apps which fall in the fun category might be better to use native approach or some cross-platform framework, that can handle all the requirements.

4.9.2 Functional apps

In functional apps the selection can be more complex. Many functional apps like online marketplaces or banking apps are used as a tool to perform some operation e.g. ordering some product or paying bills. Web app might be enough, but offering native app is also a branding question (Opinion: Native vs. mobile web app - are we missing the point?2012). In functional apps the requirements may vary lot, and the decision should be done individually for each case. E.g. when thinking applications that rely heavily on text-based content like library applications, the performance requirements do not play the biggest role. If it seems unclear, which are the most important properties the app should require, questions like what the company brand wants to achieve with the app or what the audience expects from the company when choosing the platform. Also, application context and category should be taken in to account (Opinion: Native vs. mobile web app - are we missing the point?2012).

5. Discussion and conclusion

This chapter justifies the importance of this research and describes how this research is beneficial for the software industry. It also sums up the most important findings and implications with few examples.

5.1 Benefits for the software industry

This research could be useful to companies struggling to choose which platform(s) they should choose for their upcoming app or which cross-platform technologies they should be using if they decide to rely on cross-platform approach. The decision must be made to start the development work. Companies do not want to drift in situation where they realize after the first released version, that the app cannot meet the requirements and framework used cannot manage to solve the problems. Table 1 in chapter 4 gathers the platforms and approaches together and compares them by the requirements. By looking this table, the reader should be able to find the potential option(s) for their upcoming app. Decision between native and web approaches should be easy to make because they represent the extreme opposites for most of the requirements. The decision between different cross-platform approaches might be harder to make, so more specific comparison between cross-platform tools representing three different approaches is summarized in table 2.

The interest towards this subject strived from my own short experience in software industry. Answer for question; *"what platform should we select?"* seemed to be hard to make. When diving more into the subject, I found that selecting the platform for mobile application is common problem among companies. Companies often want to produce app for all platforms, but limited recourses often appear as a decisive constraint. The purpose of this thesis was to find the important properties for mobile apps and sort out how different platforms/solutions and tools performs these properties. Providing this information, it should become clearer to create the decision what platform or solution to choose for upcoming mobile application.

5.2 Implications

Exploring previous studies and articles discussing this subject, I found series of advantages and disadvantages between native and web and apps that should be taken in concern when selecting the platform. Performance, user experience, access to hardware features, platform coverage and cost of development are seemed important properties for mobile apps (Charland & LeRoux, 2011). The problem in most cases are that native apps and web apps perform these properties with the opposite capability, so it makes hard to decide, should company produce a web app, native app or both. Due to this, there has been developed and introduced solutions like hybrid apps, interpreted apps, cross-compiled apps and progressive web apps. These reforms have made cheaper to produce apps for multiple platforms. It's still good to keep in mind that they also have their own restrictions compared to native apps, which should be considered before starting the development work.

I found also that apps can be divided in fun and functional apps (Opinion: Native vs. mobile web app - are we missing the point?2012). Fun apps, usually games, more often require many properties that native apps can handle better. With functional apps the decision should be done with more specific review. For this purpose, watching table 1

could be fast way to weigh the approaches making the decision clearer between these options. A rough estimate based on the findings would be that the more power-intensive and complex the app is, one should choose approach near the right side of the table. If app is relatively simple, perhaps needs access for location and ability to make push notifications, but the budget is rather tight PWA might be good choice. In many cases companies, addition to previous want the app to be published as a native app in app stores. If this property is considered important, hybrid app seems the most reasonable solution.

5.3 Conclusion

The main purpose of this thesis was to find answers to following questions; **RQ1**: What factors or requirements should affect the mobile application platform/approach selection? **RQ2**: How well different platforms and approaches perform on different requirements? Answering what requirements different type of applications have, was not the main purpose of the research, but also shortly discussed.

I found that important properties that should be used to compare the different platforms and approaches were; development costs, supported platforms, performance, quality of UX, sensor and device access, monetization and app maintenance (Dhillon & Mahmoud, 2015). Then I compared six different approaches for developing mobile apps. Based on prior studies and experience it is clear, that native applications perform better in performance, UX and device & sensor accessibility, than web apps (Charland & LeRoux, 2011; Liu et al., 2015). Web apps in the other hand are cheapest to produce and they cover the largest number of devices (Charland & LeRoux, 2011).

The problem of choosing between native and web app have forced industry to develop reforms and new technologies for utilizing the both sides advantages and defeating the disadvantages. Different cross-platform approaches like hybrid, interpreted and cross-compiled apps have made it cheaper to produce apps for multiple platforms (Smutny, May 2012; Xanthopoulos & Xinogalos, 2013) and PWAs (Progressive Web Apps) have increased the performance, UX and accessibility to phones hardware for web apps (Luntovskyy, 2018; Shahzad, 2017).

This thesis strived to assist the decision making between these approaches by comparing them by properties considered important by the producer and user. Comparison of cross-platform tools were made by taking look to framework documentations and previous studies comparing the frameworks, but new empiric research comparing the modern cross-platform frameworks would be useful for the future. Some of the key frameworks involved in the previous studies were discontinued and there’s not enough research material about the new frameworks replacing the discontinued frameworks.

6. References

Android developers - monetization. Retrieved 6.5.2019 from https://developer.android.com/distribute/best-practices/earn/monetization-options

Bai, J., Wang, W., Qin, Y., Zhang, S., Wang, J., & Pan, Y. (2019). BridgeTaint: A bi-directional dynamic taint tracking method for JavaScript bridges in android hybrid applications. *IEEE Transactions on Information Forensics and Security, 14*(3), 677-692. doi:10.1109/TIFS.2018.2855650

Boushehrinejadmoradi, N., Ganapathy, V., Nagarakatte, S., & Iftode, L. (2016). (2016). Testing cross-platform mobile app development frameworks. Paper presented at the *Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015,* 441-451. doi:10.1109/ASE.2015.21 Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84963801260&doi=10.1109%2fASE.2015.21&partnerID=40&md5=2158a912b2b68210985e87acdfe09530

Business models and monetization - app store - apple developer. Retrieved 6.5.2019 from https://developer.apple.com/app-store/business-models/

Charland, A., & LeRoux, B. (2011). Mobile application development: Web vs. native. *Communications of the ACM, 54*(5), 49-53. doi:10.1145/1941487.1941504

Ciman, M., & Gaggi, O. (2017). An empirical analysis of energy consumption of cross-platform frameworks for mobile development. *Pervasive and Mobile Computing, 39*, 214-230. doi:10.1016/j.pmcj.2016.10.004

CLABURN, T. (2014). Web vs. native apps: Control is what matters now. *Informationweek,* (1395), 6-7. Retrieved from http://pc124152.oulu.fi:8080/login?url=

Dalmasso, I., Datta, S. K., Bonnet, C., & Nikaein, N. (2013). (2013). Survey, comparison and evaluation of cross platform mobile application development tools. Paper presented at the *2013 9th International Wireless Communications and Mobile Computing Conference, IWCMC 2013,* 323-328. doi:10.1109/IWCMC.2013.6583580 Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84883695902&doi=10.1109%2fIWCMC.2013.6583580&partnerID=40&md5=1159bf2cf9bc06d1b24ea42bb06321d8

Dhillon, S., & Mahmoud, Q. H. (2015). An evaluation framework for cross-platform mobile application development tools. *Software - Practice and Experience, 45*(10), 1331-1357. doi:10.1002/spe.2286

Documentation - apache cordova. Retrieved 2.5.2019 from https://cordova.apache.org/docs/en/latest/

Ebone, A., Tan, Y., & Jia, X. (2018). (2018). A performance evaluation of cross-platform mobile application development approaches. Paper presented at the *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft),* 92-93.

El-Kassas, W. S., Abdullah, B. A., Yousef, A. H., & Wahba, A. (2014). (2014). ICPMD: Integrated cross-platform mobile development solution. Paper presented at the *Proceedings of 2014 9th IEEE International Conference on Computer Engineering and Systems, ICCES 2014,* 307-317. doi:10.1109/ICCES.2014.7030977 Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84946686164&doi=10.1109%2fICCES.2014.7030977&partnerID=40&md5=a84d28df2ec5e98ff7bc0e51c90c5b02

El-Kassas, W. S., Abdullah, B. A., Yousef, A. H., & Wahba, A. M. (2017). Taxonomy of cross-platform mobile applications development approaches. *Ain Shams Engineering Journal, 8*(2), 163-190. doi:10.1016/j.asej.2015.08.004

Fortunato, D., & Bernardino, J. (2018). (2018). Progressive web apps: An alternative to the native mobile apps. Paper presented at the *Iberian Conference on Information Systems and Technologies, CISTI, , 2018-June* 1-6. doi:10.23919/CISTI.2018.8399228 Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-85049887518&doi=10.23919%2fCISTI.2018.8399228&partnerID=40&md5=27000cf86a03e4cfb2b9acd6a9a6e91b

Frankston, B. (2018). Progressive web apps [bits versus electrons]. *IEEE Consumer Electronics Magazine, 7*(2), 106-117. Retrieved from https://oula.finna.fi/PrimoRecord/pci.ieee_s8287006

Getting started · react native. Retrieved 2.5.2019 from https://facebook.github.io/react-native/

Google play. (2019). Retrieved 8.5.2019 from https://play.google.com/store/apps

Gronli, T. -., Hansen, J., Ghinea, G., & Younas, M. (2014). (2014). Mobile application platform heterogeneity: Android vs windows phone vs iOS vs firefox OS. Paper presented at the *Proceedings - International Conference on Advanced Information Networking and Applications, AINA,* 635-641. doi:10.1109/AINA.2014.78 Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84903833463&doi=10.1109%2fAINA.2014.78&partnerID=40&md5=05ad6034b1e6b9b7d6e8eb51a2a86c8a

Han Rebekah Wong, S. (2012). Which platform do our users prefer: Website or mobile app? *Reference Services Review, 40*(1), 103-115. Retrieved from https://oula.finna.fi/PrimoRecord/pci.emerald_s10.1108%2F00907321211203667

Kho, N. D. (2018). Everything you need to know about progressive web apps. *EContent, 41*(2), 20-24. Retrieved from http://pc124152.oulu.fi:8080/login?url=

Kim, B. (2013). Responsive web design, discoverability, and mobile challenge. *Library Technology Reports, 49*(6), 29-30. Retrieved from http://pc124152.oulu.fi:8080/login?url=

Liu, Y., Liu, X., Ma, Y., Liu, Y., Zheng, Z., Huang, G., & Blake, M. B. (2015). (2015). Characterizing RESTful web services usage on smartphones: A tale of native apps and web apps. Paper presented at the *Proceedings - 2015 IEEE International Conference on Web Services, ICWS 2015,* 337-344. doi:10.1109/ICWS.2015.53 Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-

84956680073&doi=10.1109%2fICWS.2015.53&partnerID=40&md5=00e0b9ea64c0a5276529083d1b26cefd

Luntovskyy, A. (2018). (2018). Advanced software-technological approaches for mobile apps development. Paper presented at the *14th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering, TCSET 2018 - Proceedings, , 2018-April* 113-118. doi:10.1109/TCSET.2018.8336168 Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-85047561811&doi=10.1109%2fTCSET.2018.8336168&partnerID=40&md5=0a2226ab9b1df0bfad165eda5b9cf6ea

Ma, Y., Liu, X., Liu, Y., Liu, Y., & Huang, G. (2018). A tale of two fashions: An empirical study on the performance of native apps and web apps on android. *IEEE Transactions on Mobile Computing, 17*(5), 990-1003. Retrieved from http://pc124152.oulu.fi:8080/login?url=

Malavolta, I., Ruberto, S., Soru, T., & Terragni, V. (2015). (2015). Hybrid mobile apps in the google play store: An exploratory investigation. Paper presented at the *2015 2nd ACM International Conference on Mobile Software Engineering and Systems,* 56-59. doi:10.1109/MobileSoft.2015.15

Martinez, M., & Lecomte, S. (2017). (2017). Towards the quality improvement of cross-platform mobile applications. Paper presented at the *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft),* 184-188. doi:10.1109/MOBILESoft.2017.30

Nakajima, S. (2012). Apps and the mobile internet the battle of the platforms: Both native and web apps.(86), 209-215. Retrieved from http://pc124152.oulu.fi:8080/login?url=

Nunkesser, R. (2018). (2018). Beyond web/native/hybrid: A new taxonomy for mobile app development. Paper presented at the *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft),* 214-218.

Opinion: Native vs. mobile web app - are we missing the point? (2012, *New Media Age (Online),* , n/a. Retrieved from https://search.proquest.com/docview/1022098863?accountid=13031

Palmieri, M., Singh, I., & Cicchetti, A. (2012). (2012). Comparison of cross-platform mobile development tools. Paper presented at the *2012 16th International Conference on Intelligence in Next Generation Networks, ICIN 2012,* 179-186. doi:10.1109/ICIN.2012.6376023 Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84871874191&doi=10.1109%2fICIN.2012.6376023&partnerID=40&md5=98f50b7842b8eea9c86fdd5738864e08

Shahzad, F. (2017). (2017). Modern and responsive mobile-enabled web applications. Paper presented at the *Procedia Computer Science, , 110* 410-415. doi:10.1016/j.procs.2017.06.105 Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-85028656136&doi=10.1016%2fj.procs.2017.06.105&partnerID=40&md5=035d70ad3db4a2a0677338fee4c83042

Smutny, P. (May 2012). (May 2012). Mobile development tools and cross-platform solutions. Paper presented at the 653-656. doi:10.1109/CarpathianCC.2012.6228727 Retrieved from https://ieeexplore.ieee.org/document/6228727

Titanium SDK - appcelerator platform - appcelerator docs. Retrieved 2.5.2019 from https://docs.appcelerator.com/platform/latest/#!/guide/Titanium_SDK

Turgeman, L., Smart, O., & Guy, N. (2019). Unsupervised learning approach to estimating user engagement with mobile applications: A case study of the weather company (IBM). *Expert Systems with Applications, 120*, 397-412. doi:10.1016/j.eswa.2018.11.037

Xamarin documentation - xamarin. Retrieved 2.5.2019 from https://docs.microsoft.com/en-us/xamarin/

Xanthopoulos, S., & Xinogalos, S. (2013). (2013). A comparative analysis of cross-platform development approaches for mobile applications. Paper presented at the 213–220. doi:10.1145/2490257.2490292 Retrieved from http://doi.acm.org/10.1145/2490257.2490292